Appendix A

Energy efficiency of error correction for wireless communication

Since high error rates are inevitable in the wireless environment, energy efficient error control is an important issue for mobile computing systems. When error-correction mechanisms are implemented on general-purpose processors the power consumption that is required to perform the error-correction mechanism can be significant. To illustrate this we have studied two different error correction mechanisms with different characteristics and capabilities, i.e. EVENODD and Reed-Solomon[1].

A.1 Introduction

Traditionally, network protocols have been designed around the conventional metrics of throughput and latency. However, a proper design of these protocols offers many opportunities for optimising the design metric that is more relevant to battery operated devices: the amount of energy consumed per useful user level bit transmitted across the wireless link. Since high error rates are inevitable to the wireless environment, *energy-efficient error-control* is an important issue for mobile computing systems. This includes energy spent in the physical radio transmission process, as well as energy spent in computation, such as signal processing and error control at the transmitter and the receiver.

In communication systems forward error correcting (FEC) codes are used to protect packets of data that are transmitted over some network. Error-control mechanisms traditionally trade-off complexity and buffering requirements for throughput and delay [4][11][12].

Error-correcting codes are generally applied at several layers in the communication protocol stack. Error-control at the lower layers (physical, data link layer) is often implemented with dedicated hardware and embedded software. We will concentrate on error correction mechanisms for the higher protocol layers, which are mostly

[1] Major parts of this chapter have been presented at the *IEEE Wireless Communications and Networking Conference 1999* [7].

implemented in software. Dedicated hardware may be used to implement (parts) of the error control (as described in Chapter 5), though in a system, which requires the flexibility to alter error-control schemes on a stream by stream basis, a software solution may be preferable.

At these layers the error correction mechanism operates on relative large blocks. Generally, block codes such as Bose, Chaudhuri and Hockuenghem (BCH) and Reed-Solomon codes require a decoder capable of performing arithmetic operations in finite fields [14]. A comparison between application-specific integrated circuit (ASIC), FPGA, and digital signal processing (DSP) implementations of the decoder shows that the performance of FPGA-based designs lean more toward that of ASICs, but retain flexibility more like DSPs [3][6]. Unfortunately, good VLSI designs for codes using BCH or Reed-Solomon codes do not map well to FPGAs [1]. A code that does not require finite-field arithmetic, is the *EVENODD* code [2]. The EVENODD code was originally designed for a system of redundant disks (RAID).

When error-correction mechanisms are implemented on general-purpose processors the power consumption that is required to perform the error-correction mechanism can be significant. We have studied a software implementation of the EVENODD error correcting mechanism, and compared it with an implementation of the Reed-Solomon mechanism.

The total energy consumption per useful bit will depend both on the energy of transmission and the energy of redundancy computation. We will show that the computational cost associated with FEC cannot be ignored, constituting a significant portion of the overall energy cost. Furthermore, the trend has been toward smaller communication cells, e.g. with the size of an office room, thus requiring lower transmit power. The ratio of computational to transmit power under these circumstances is therefore only likely to increase.

A.1.1 The encoding packet model

The basis for most currently designed wireless systems is packet switching, which manages data transfer in blocks (*packets*) that contain multiple symbols (or bits). The size of a packet is in principal not related to the actual amount of data transmitted over the channel in a MAC frame. Errors are assumed to be detected by some detection technique (e.g. by using Cyclic Redundancy Check (CRC) data), and the whole packet will be discarded. The residual channel characteristic after the physical and link layer processing is then based on *erasures*, i.e. missing packets in a stream [17]. Figure 1 shows a graphical representation of the error correction mechanism. The sender collects a number of *source data packets* in a buffer. When the buffer is full, the data is encoded, and the encoded data is transmitted. The receiver is able to reconstruct the original data from a subset of the encoded data, and so can allow the erasure of some packets.

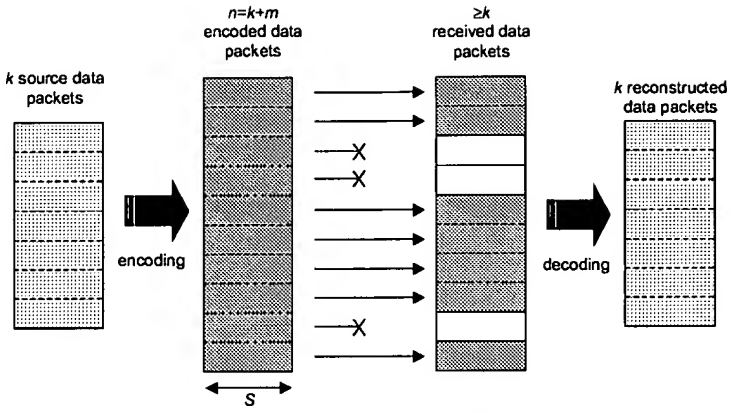

Figure 1: Graphical representation of error correction.

We will denote the number of source packets as k, the packet size as S, the number of redundant packets m, and the number of encoded packets as n. Such a code is called an (n,k) code and allows the receiver to recover from m $(=n-k)$ losses in a group of n encoded packets. This structure can be seen as an (S) x $(k + m)$ array in which the columns represent a packet of length S, the first k columns represent the source data packets, and the last m columns represent the redundant packets. All packets together build up one *frame*. Figure 2 gives a graphical representation of this scheme.

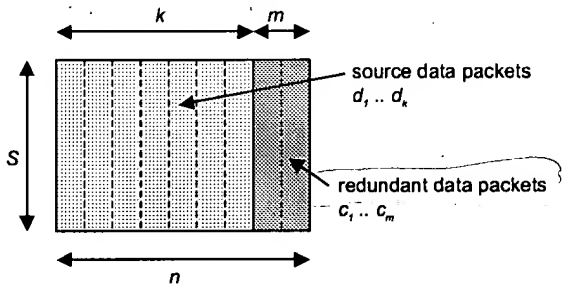

Figure 2: Representation of an encoded frame.

A general technique for tolerating m simultaneous failures with m redundant packets is a technique based on *Reed-Solomon* coding [14]. This technique requires computation over finite fields and results in a complex implementation. An alternative might be a scheme like EVENODD that only requires simple exclusive-OR operations and that it is able to tolerate two erasures. In the following section we will give an overview of the EVENODD and Reed-Solomon coding and determine the energy efficiency of these mechanisms.

We define the *energy efficiency e* as the amount of data processed divided by the energy that is consumed to process that data:

$$e = \frac{\textit{Amount of data}}{\textit{Energy consumed to process the data}} \quad [J^{-1}] \qquad (1)$$

One can view this as the inverse of the cost (in terms of energy) of calculating the redundancy to be transmitted over a channel.

A.1.2 Reed-Solomon coding

The Reed-Solomon coding scheme is an (n,k) code. There are three main aspects involved with the Reed-Solomon algorithm: the use of the Vandermonde matrix to calculate the redundant packets with simple matrix arithmetic, the use of Gaussian elimination to recover from failures, and the use of Galois fields to perform arithmetic [16].

A major concern is that the domain and range of our computations are binary words of a fixed length w. Since practical algebra implementation does not use infinite precision real numbers, we must perform addition and multiplication over a *finite field* of more than $k + n$ elements. Fields with $q = p^w$ elements, with p prime and $w > 1$ are called extension fields or Galois Fields denoted as $GF(p^w)$. Operations on extension fields are simple in the case $p = 2$. The elements of $GF(2^w)$ are integers from zero to $2^w - 1$. Addition and subtraction of $GF(2^w)$ are simple exclusive-OR operations. Multiplication and division are more complex and require two mapping tables, each of length 2^w. These two tables map an integer to its logarithm and its inverse logarithm in the Galois field. A table for the multiplication can be used as well if the number of field elements is not too large. Note that a multiplication in $GF(2^8)$ already requires a 64 kB lookup table! However, a Reed-Solomon coding implementation that is not parameterised (i.e. n and k are fixed) can be implemented much more efficient [9].

Energy efficiency of the Reed-Solomon algorithm

The *encoding overhead* depends on the number of source packets k, on the number redundant packets m $(= n - k)$ and on the size S of a packet. The encoder requires k source data packets to produce each encoded packet, and thus the encoding overhead to process k source packets is $O((n-k).k.S)$. Therefore, an approximation of the energy efficiency of encoding e_{rse} equals:

$$e_{rse} = E_{rse} \frac{k}{(n-k)\,k} = E_{rse} \frac{1}{m} \qquad (2)$$

in which E_{rse} is the efficiency for encoding with Reed-Solomon. The value is determined by the implementation and is dependent on the packet size S.

The *decoding overhead* is more complicated as it involves two parts: the Gaussian elimination, and the reconstruction. This requires a matrix inversion to be performed

once, and then a matrix multiplication for each reconstructed packet which is maximal m. Although the matrix inversion requires $O(k.(n-k)^2)$ operations per k packets, the cost of matrix inversion becomes negligible for reasonable sized packets (matrix inversion is not required for a non-parameterised implementation with $m \leq 2$ like EVENODD). In the experiments this will be shown clearly. The matrix multiplication requires $O(k)$ operations for each reconstructed data item, or a total of $O((n-k).k.S)$ operations per block of k packets. So, if we assume the number of reconstructed packet to be equal to $(n-k)$ then an approximation of the energy efficiency of decoding e_{rsd} equals:

$$e_{rsd} = E_{rsd} \frac{k}{(n-k).k} = E_{rsd} \frac{1}{m} \quad (3)$$

in which E_{rsd} is the efficiency for decoding with Reed-Solomon. The value is determined by the implementation and is dependent on the packet size S.

A.1.3 EVENODD coding

The EVENODD coding scheme is an $(k+2,k)$ code. It was originally meant for tolerating two failures in RAID architectures, but we will show that it is also suitable in communication systems. The basic scheme requires the number of source packets k to be a prime number. If we want to use a non-prime number for k, then we can take the next prime following the required k, and assume the extra imaginary packets to contain zeros. The packet size S is for simplicity restricted to contain $(k-1)$ symbols. This restriction is not hard too because a symbol can be of any size, and also because we can introduce imaginary symbols to fill the columns to the required size.

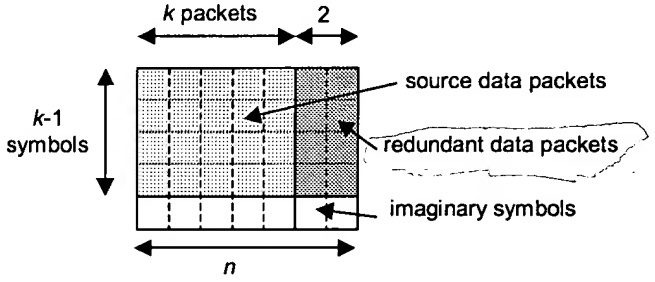

Figure 3: Basic EVENODD frame.

So, the packet model can be considered as an $(k-1)$ x $(k+2)$ array, k a prime number, such that the symbol d_{ij}, $0 \leq i \leq (k-2)$, $0 \leq j \leq (k-1)$, is the i-th symbol (row) in the j-th packet (column). The last two packets (k and $k+1$) are the redundant packets. One imaginary row (k-1) is added containing zeros.

Data encoding

There are two types of redundancy: *horizontal* redundancy and *diagonal* redundancy. The redundant value of each is stored in a redundant packet. The value of the horizontal redundancy (stored in packet k) is the exclusive-OR of packets 0, 1, …, $k-1$. This is thus exactly the same as with simple parity encoding. Packet (k+1) carries a diagonal redundancy. This is calculated using the exclusive-OR of the diagonals of the matrix and P. P is calculated via the exclusive-OR of a special diagonal. So, for example the first redundant symbol in redundant packet $k+1$, denoted as $d_{0,k+1}$, is calculated with: $d_{0,k+1} = P \oplus d_{0,0} \oplus d_{k-1,1} \oplus d_{k-2,2} \oplus \ldots \oplus d_{1,k-1}$. Since the source packet matrix is an $(k-1)$ x (k) matrix, one diagonal is not calculated. This diagonal (formed by the indices (k-1,0), (k-2,1), (k-3,2), …, (0,k-1)) is used to determine the value of P.

	0	*1*	*2*	*3*	*k-1*	*k*	*k+1*
0	1	0	1	1	0	1	0
1	0	1	1	0	0	0	0
2	1	1	0	0	0	0	1
k-2	0	1	0	1	1	1	0

Figure 4: EVENODD coding example for k=5.

An example of an encoded frame with symbols of one bit is shown in Figure 4. Notice that (without the imaginary row k-1)

$P = d_{3,1} \oplus d_{2,2} \oplus d_{1,3} \oplus d_{0,4}$. , and e.g. $d_{2,6}$ is obtained as follows: $d_{2,6} = P \oplus d_{2,0} \oplus d_{1,1} \oplus d_{0,2} \oplus d_{3,4}$.

Data recovery

Data encoded with the EVENODD scheme is able to recover maximal two packet erasures. This equivalents to Reed Solomon encoding with n-k=2. Reconstruction when only one packet is erased (and assuming it is not a redundant packet) is simple as the missing packet can be retrieved using the exclusive-OR of the packets. When two packets i and j, $0 \le i < j \le k+1$, are erased, then the decoding scheme is more complicated, but still requires only exclusive–OR operations. Note that recovering is also possible for finer grained erasures: i.e. not all erasures need to be in the same two packets. Depending on the topology of the symbol erasures up to $2(k-1)$ *symbol erasures* can be restored [8]. A similar effect can be reached with Reed-Solomon coding when interleaving is used.

Energy efficiency of EVENODD coding

Encoding for the horizontal parity packet requires k exclusive-OR operations to be performed on each data symbol with size s. The diagonal parity requires $k+1$ exclusive-OR operations, including the calculation of P. This makes the total encoding complexity

$O(2k+1)$. The amount of data encoded is $(k-1).s.k$. Using the packet size $S=(k-1).s$ we get $S.k$. Therefore, the energy efficiency e_{eve} is:

$$e_{eve} = E_{eve} \frac{k}{2k+1} \qquad (4)$$

in which E_{eve} is the implementation dependent efficiency for encoding EVENODD. The value of E_{eve} is determined by the specific implementation in either hardware or software and is dependent on the packet size S.

The *decoding overhead* is dependent on the number of erased packets, which packets are erased (i.e. whether redundant packets are involved or not), the number of source packets k, and on the size s of a symbol. We will only deal with the complexity of the erasure of two data packets as this is the most most complex. This case has three main steps. First, calculating the diagonal parity P requires $O(2 . S)$ exclusive-OR operations. Then two syndromes are calculated, requiring $O(S . (k-1))$ and $O(S . k)$ XOR operations. Finally the reconstruction takes another $O(2 . S)$ XOR operations. This makes the total decoding complexity $O((2k + 3) S)$, which are basically all XOR operations. When E_{eod} is the efficiency for decoding EVENODD then the energy efficiency equals:

$$e_{eod} = E_{eod} \frac{k}{2k+3} \qquad (5)$$

The value of E_{eod} is determined by the specific implementation in either hardware or software and is dependent on the packet size S.

A.2 Implementation and results

A.2.1 Software implementation

In the next sections we show the results of a software implementation on a general-purpose processor of both error correction mechanisms. Such an implementation is the most flexible solution and can adapt its algorithm and its parameters very quickly to changing environments. Adaptive error correction has shown to be much better than a single code scheme in terms of utilised bandwidth and in terms of a profit function which combines the bandwidth utilised and the deadline miss rate [5].

We are aware of the fact that a software implementation is not the most energy-efficient solution, and might not provide enough performance. However, there exist many applications and systems that do not need high performance and cannot use the capabilities and advantages of dedicated hardware. For example, a notebook computer that lacks such dedicated hardware can also benefit from an energy-efficient solution, even if it is not the most optimal implementation.

A software implementation has a number of specific advantages compared to a hardware solution:

- The use of a microprocessor allows *very rapid adaptations* to varying error conditions (burst size, frequency) and required QoS from applications. The adaptation to perform can be applying another error-control scheme, or adapting some parameters of the error-control scheme.
- A software implementation allows us to experiment with a large set of error-control schemes, and experience in 'real life' how applications behave. When we have a good feeling of the behaviour of the schemes, then we could compose a subset of error-control schemes that is suitable to be implemented in hardware, either in an FPGA, a DSP, or a custom chip.
- The error control can easily and efficiently be embedded in various layers of the communication protocol where the data is buffered anyway. With a good engineered and well-integrated error correction mechanism little extra overhead is expected.
- A standard processor also allows the use of relatively large memories, and thus allows for much larger block lengths than standard custom chips (that typically allow a block length of up to 255 bytes [13]). In a wireless office environment burst errors of 1 to 100 ms can be expected. To handle these large erasures at relatively high speed (say 2 Mb/s), a large block size is needed.

We will assume that there is a linear relation between the energy consumed by the algorithm and the amount of time needed for the processor to do its calculations. Although this assumption introduces some inaccuracy (for example because this does not incorporate possible energy management mechanisms of the processor, the energy consumption of the memories being used for buffering, and the energy consumption of other parts of the computer system that also needs to be active), this still gives a good indication of reality [9]. Both implementations are written in *C* and are portable across many platforms. The implementation of the Reed-Solomon coding is flexible, it can be used for arbitrary n and k. The measurements were performed on a Toshiba 220CS notebook that has a Pentium Pro 133 processor and runs Windows 95. The results can only be used as a reference, since the actual performance depends on items like memory speed, cache size, quality of the compiler, operating system, etc. The code is written straightforward, and uses the most obvious optimisations only (like the use of a multiply lookup table for Reed-Solomon coding). Handcrafted code that makes good use of the specific features of the processor (like registers and the use of special instructions) might achieve significant speedups.

Using the timing measurements we can calculate the costs χ (in µs/byte) to encode and decode one byte. The efficiency E can then be defined as $1/\chi$, which is independent from the actual power consumption of the processor on which the coding was performed. The energy efficiency e incorporates the power consumption P of the processor, thus $e = 1 / \chi P$ (in Joule^{-1}).

A.2.2 EVENODD coding implementation

The data model that we have used in our implementation of the EVENODD coding resembles the basic model in which an imaginary 0-row is added to simplify the implementation [1]. So, we use a (k) x (k + 2) array, k a prime number, of symbols with size s. Each column represents a packet. The last two columns (k and k + 1) are the redundant columns. The symbols can be of any size, but normally are a multiple of a byte. Note that the values of the efficiency E_{eoe} and E_{eod} are independent from the power consumption of the processor.

Figure 5 and Figure 6 show the characteristics of the efficiency E_{eoe} and E_{eod} of Equation (4) and Equation (5) versus the number of source packets k, for various values of the symbol size s.

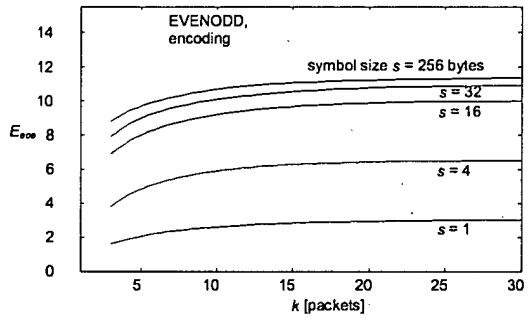

Figure 5: Efficiency E_{eoe} vs. k as a function of symbol size s.

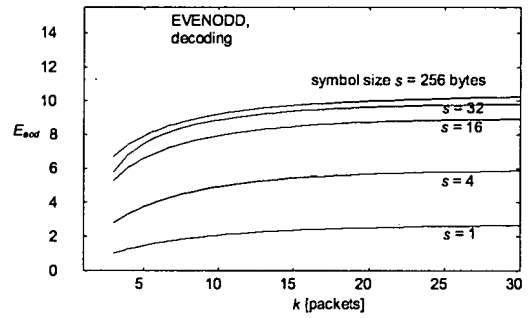

Figure 6: Efficiency E_{eod} vs. k as a function of symbol size s.

The effect of the implementation overhead gets less when the packet size over which the code has to work is enlarged, because it will be amortised over more data. A better performance is also reached due to the effect of caching, since the same data is used several times. In the constant region the encoding time on the Toshiba 220 CS is approximately 88 µs/kByte and the decoding time approximately 96 µs/kByte.

A.2.3 Reed-Solomon coding implementation

Our code of the Reed-Solomon coding is based on an implementation from Rizzo, Karn and others [18]. The data model that we have used in our measurements is an (S) x (k + (n-k)) array of symbols with size s. Each column represents a packet with size S. The last n-k columns are the redundant columns. The code supports $GF(2^w)$, for any w in the range of 2..16. In the measurements we have used w = 8. This gives the maximum efficiency because most operations can be executed using lookup tables [17]. So, the symbol size s in our measurements will be one byte. We chose S to be multiples of ATM cells sizes (53 bytes). We have used a lookup table for the multiply operations.

Figure 7 shows the characteristics of the efficiency E_{rse} and E_{rsd} (of Equation (2) and (3)) versus k, for various values of S. The energy efficiency of encoding is hardly influenced by the packet size or the number of source packets; therefore only one graph is shown in the figure for encoding. Encoding is already stable for small values of k and for all packet sizes.

Decoding is more influenced by the packet size. This is mainly caused by the cost of matrix inversion which cost $O(k \cdot l^2)$, where l is the number of packets which must be recovered (which we assume to be equal to n-k). The influence is small for packet sizes greater or equal to 8 ATM cells only.

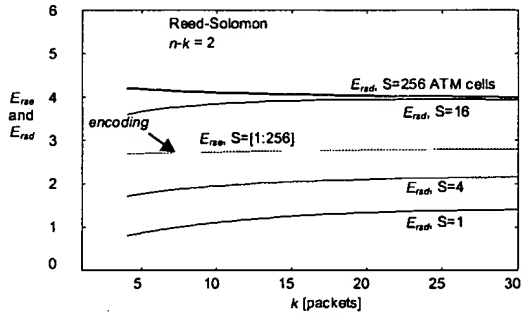

Figure 7: Efficiency vs. number of source packets as a function of packet sizes.

In the constant region the encoding time is approximately 365 µs/kByte and the decoding time is approximately 257 µs/kByte.

A.2.4 Comparison

We can compare the implementations of the EVENODD and the Reed-Solomon mechanisms for n-k=2. Both implementations reach a constant performance with constant overhead at small values of k and for small data sizes. We calculated the energy efficiency e using the power consumption of the Pentium Pro 133 processor, which is approximately 14 W [15]. To summarise we have the following results:

mechanism	speed [µs/kB]	E	e [µJ^{-1}]	Minimal k	Minimal data size
EVENODD encoding	88	11.6	0.83	5	symbol: 32 bytes
EVENODD decoding	96	10.6	0.76	5	symbol: 32 bytes
Reed-Solomon encoding	365	2.8	0.20	5	packet: 1 ATM cells
Reed-Solomon decoding	257	3.9	0.28	5	packet: 16 ATM cells

Table 1: Characteristics of error correcting codes for n-k=2.

The efficiency of the implementation of the mechanisms will in general be a bit better when larger k and/or symbol sizes are used. The used implementation of the Reed-Solomon encoding is about four times as inefficient as EVENODD. Decoding is more than two times as inefficient. The minimal size of a data item (either a symbol or a packet) depends on the choice of the packet size for EVENODD. When the packet size is chosen to be one column (just like our Reed-Solomon implementation), then the minimum size of a packet for EVENODD equals 32(k-1). The minimum packet size for EVENODD is thus smaller than for Reed-Solomon for approximately k<26. E.g. when k=7, then the minimal packet size for EVENODD equals just more than 4 ATM cells, which is much less than the minimum of 16 cells for Reed-Solomon coding. Note that a non-parameterised implementation of the Reed-Solomon code can be more efficient and has less initial overhead [9].

A.2.5 A minimal communication system

Error correction mechanisms for wireless communication involve computational overhead and communication overhead at both the transmitter and the receiver side. This is overhead in time, but also overhead in energy consumption. In our context we mainly focus on the *energy overhead*. The overhead is composed out of two elements, the encoding overhead and the communication overhead.

In the previous sections we have investigated the *computational* energy efficiency of two error correction mechanisms. We will now consider the *energy efficiency of a system* in which also the energy consumption of the communication interface is incorporated. We will only consider the data that is actually transmitted, and not incorporate additional costs involved with the wireless interface like turning 'on' and 'off' the transceiver, sending extra control data, etc. These matters are dealt with in e.g. the medium access control layer. A more precise analysis would require these costs to be incorporated as well. However, these costs are dependent on the underlying protocols and operating system, and the energy savings capabilities of the system. So, to have a clean comparison we will only use in our analysis the energy needed for the actual data transfer.

The *communication overhead* mainly depends on the number of additional bits that are transmitted. The number of redundant bytes equals the number of redundant packets m multiplied by the packet size S, and thus the total communication overhead of k source packets is $O(mS)$. So the communication energy efficiency of transmitting an (n,k) redundant code equals:

$$e_{com} = E_{com} \cdot k / m \qquad (6)$$

in which E_{com} is the energy-efficiency factor that is determined by the energy consumption of the wireless interface.

As an example we will determine the energy efficiency of a small system consisting of a WaveLAN PCMCIA card as wireless communication device and a Pentium Pro 133 MHz as general-purpose processor (the same processor as used in our experiments). We will compare the efficiency using a rating ρ that indicates the amount of energy consumed to process one byte, using:

$$\rho = \textit{time to process 1 byte [s]} \cdot \textit{required power [W]} \qquad (7)$$

The WaveLAN 2.4 GHz modem interface consumes approx. 1800 mW when transmitting [19]. The data transfer-rate is 2Mb/s. One byte takes thus 4 µs to process, which results in an energy consumption of ρ=7.2 µJ/byte. The Pentium Pro 133 processor takes 14 W [15]. As an example we will now compare this with the energy consumption to encode data with EVENODD. The time needed to encode 1 kB of data using the EVENODD mechanism is 88 µs, so one byte takes 88/1024 µs, resulting in an energy consumption of ρ=1.2 µJ/byte. The energy-efficiency ratio between encoding with EVENODD and communication thus equals 7.2 / 1.2 = 6.0.

This shows that when the power consumption of the wireless interface is relatively high, then it is worthwhile to use adaptive error control that tries to minimise the amount of data transmitted over the wireless channel. However, if the energy consumption of the wireless interface is relatively low compared to the power consumption required to implement the error control, then it might be more effective to utilise an error coding that is optimised for worst case conditions and does not need the control-loop.

A.3 Conclusion

When error-correction mechanisms are used for wireless systems, a major design criterion should be the energy efficiency of a mechanism. *Adaptable error correction*, that adapts its parameters and scheme according to the error-rate and required QoS, can be used to trade-off between performance and cost, including the required energy consumption.

We have shown that the power consumption that is required to perform the error-correction mechanism can be significant. To illustrate this we have studied two implementations of different error correction mechanisms with different characteristics

and capabilities, i.e. EVENODD and Reed-Solomon. We have identified that the choice of whether to apply adaptive error control depends on the energy efficiency of the error control, and on the energy consumption of the wireless interface. Adaptive error control is effective only when the energy consumption of error control to process a certain amount of data is lower than the energy consumption required to transmit this amount over the wireless channel. When the energy consumption of error control and wireless communication are almost equal, then the extra complexity and the required feedback from the receiver to the transmitter must be carefully incorporated in the decision to use adaptive error control.

The implementations of these mechanisms on a general-purpose processor show that they already reach constant performance and constant energy efficiency for small values of k and for small data sizes. The Reed-Solomon code is attractive because it is the most general technique capable of tolerating n-k simultaneous failures. The code rate can be defined fine-grained. However, this flexibility makes the encoding about four times less energy efficient than EVENODD (due to the introduced complexity and the requirement of computations in the finite field). The EVENODD mechanism on the other hand is rather course-grained. It can sustain two packet erasures, or, more generally, it allows the reconstruction of up to $2(k\text{-}1)$ erased *symbols* (and not only packet erasures as with Reed-Solomon without interleaving). This ability gives EVENODD an inherent adaptability, since it can efficiently tolerate variable burst error rates using the same code rate. This increases its flexibility and gives a further reduction in energy consumption for decoding when the burst-error size is smaller than a whole packet.

References

[1] Ahlquist G.C., Rice M., Nelson B.: "Error control coding in software radios: an FPGA approach", *IEEE Personal Communications,* August 1999, pp. 35-39, 1999.

[2] Blaum M., et al.: "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures", *IEEE Transactions on computers*, Vol. 44, No 2, pp. 192-201, February 1995.

[3] Bowers H., Zhang H.: "Comparison of Reed-Solomon codec implementations", *Technical rep. UC Berkeley*, http://infopad.eecs.berkeley.edu/~hui/cs252/rs.html.

[4] Cho, Y.J., Un, C.K.: "Performance analysis of ARQ error controls under Markovian block error pattern", *IEEE Transactions on Communications.*, Vol. COM-42, pp. 2051-2061, Feb-Apr. 1994.

[5] Elaoud, M, Ramanathan, P.: "Adaptive Use of Error-Correcting Codes for Real-time Communication in Wireless Networks", *proceedings IEEE Infocom'98*, pp. 548-555, March 1998.

[6] Goslin G.R.: "Implement DSP functions in FPGAs to reduce cost and boost performance", EDN magazine, 1996, http://www.ednmag.com/reg/1996/101096/21df_05.htm.

[7] Havinga P.J.M.: "Energy efficiency of error correction on wireless systems", *proceedings IEEE Wireless Communications and Networking Conference (WCNC'99)*, September 1999

[8] Havinga, P.J.M.: "Energy efficiency of error correcting mechanisms for wireless communication", *CTIT Technical Reports 1998*, TR-CTIT 98-19, September 1998, the Netherlands.

[9] Krol Th., personal communication.

[10] Lettieri P., Schurgers C., Srivastava M.B.: "Adaptive link layer strategies for energy efficient wireless networking", ACM WINET, 1999.

[11] Lin, S., Costello, D.J., Miller, M.: "Automatic-repeat-request error-control schemes", *IEEE Comm. Magazine*, v.22, n.12, pp. 5-17, Dec 1984.

[12] Liu, H., El Zarki, M.: "Delay bounded type-II hybrid ARQ for video transmission over wireless networks", *proceedings Conference on Information Sciences and Systems*, Princeton, March 1996.

[13] "L64711/12/13/14 Reed-Solomon Encoders/decoders", *LSI logic*, http://www.lsilogic.com/products/unit5_7d.html.

[14] MacWilliams, F.J., Sloane, N.J.A.: "The theory of error-correcting codes", *North-Holland Publicing Company*, Amsterdam, 1977.

[15] "Pentium Pro processors, Product overview", http://developer.intel.com/design/pro.

[16] Plank, J.S.: "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems", *Software, practice & experience*, 27(9), Sept 1997, pp. 995-1012.

[17] Rizzo, L.: "Effective Erasure Codes for Reliable Computer Communication Protocols", *ACM Computer Communication Review*, Vol. 27- 2, pp. 24-36, April 97.

[18] Rizzo, L., sources for an erasure code based on Reed-Solomon coding with Vandermonde matrices. Available at http://www.iet.unipi.it/~luigi/vdm.tgz.

[19] "WaveLAN/PCMCIA network adapter card", http://www.wavelan.com/support/libpdf/fs-pcm.pdf.